An Efficient Constraint Grammar Parser based on Inward Deterministic Automata

Yli-Jyrä, Anssi Mikael

Northern European Association for Language Technology
2011-11-17

Yli-Jyrä , A M 2011 , An Efficient Constraint Grammar Parser based on Inward Deterministic Automata . in E Bick , K Hagen , K Müürisep & T Trosterud (eds) , Proceedings of the NODALIDA 2011 Workshop Constraint Grammar Applications . vol. 14 (2011) , NEALT Proceedings Series , vol. 14 (2011) , Northern European Association for Language Technology , Tartu University Library, Tartu, Estonia , pp. 50-60 , Workshop in Constraint Grammar Applications in conjunction with NoDaLiDa 2011, Riga, Latvia , Riga , Latvia , 11/05/2011 . < http://dspace.utlib.ee/dspace/handle/10062/19302 >

http://hdl.handle.net/10138/28875

publishedVersion

An Efficient Constraint Grammar Parser based on Inward Deterministic Automata

Anssi Yli-Jyrä
The Department of Modern Languages, PO Box 24, 00014 University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Abstract

The paper reconceptualizes Constraint Grammar as a framework where the rules refine the compact representations of local ambiguity while the rule conditions are matched against a string of feature vectors that summarize the compact representations. Both views to the ambiguity are processed with pure finite-state operations. The compact representations are mapped to feature vectors with the aid of a rational power series. This magical interconnection is not less pure than a prevalent interpretation that requires that the reading set provided by a lexical transducer is magically linearized to a marked concatenation of readings given to pure transducers. The current approach has several practical benefits, including the inward deterministic way to compute, represent and maintain all the applications of the rules in the sentence.

1 Introduction

Constraint Grammar (CG) (Karlsson et al., 1995) is a text parsing method with benefits over statistical methods: a low memory footprint, run-time speed, linguistic detail, data bootstrapping, incremental development, and applicability to linguist's needs. Despite its wide use, the common understanding about its algorithms is still shallow. The current work attempts to reduce this gap.

1.1 The Background

The dawn of CG was marked by a number of related developments. Some resource sensitive parsers (Marcus, 1980; Krauwer and des Tombe, 1981; Church, 1988; Blank, 1989) had started to simplify over the parsers based on augmented transition networks (Woods, 1970). The Taggit program (Greene and Rubin, 1971; according to Tapanainen 1999) was an early context-dependent tagger. Some generative grammars were related to automata and local constraints both in syntax (Peters and Ritchie, 1969; Joshi and Levy, 1982), and in phonology (Johnson, 1972; Koskenniemi, 1983). Their constraints were similar to local grammars that describe word chain characteristics (Gross, 1968; Maurel, 1989; Mohri, 1994). The systems of hard constraints gave rise to consistency enforcing methods (Huffman, 1971; Barton, Jr., 1986; Maruyama, 1990).

The CG framework (Karlsson et al., 1995) applies disjunctively ordered rules iteratively to implement a system of soft constraints. Some CG parsers have been described (Karlsson, 1990; Tapanainen, 1996; Graña et al., 2003; Didriksen, 2010; Peltonen, 2011; Hulden, 2011).

The several later parsing methods bear similarities to CG. These include Finite-State Intersection Grammar (FSIG) (Koskenniemi et al., 1992), cascade parsing and chunking (Joshi and Hopely, 1996; Abney, 1991; Grefenstette, 1999), replace rule sequences (Karttunen, 1997; Aït-Mokhtar and Chanod, 1997), lenient composition (Karttunen, 1998), voting constraints (Oflazer and Tür, 1997), error-driven parsing (Brill, 1992; Lager, 2001), bimachines (Roche, 1994; Skut et al., 2004; Peikov, 2006), iterated finite transducers (Roche, 1997b; Bordihn et al., 2006), restarting automata and contextual grammars (Plátek et al., 2003; Jurdziński et al., 2005), and logic programming (Lindberg and Eineborg, 1998; Lager and Nivre, 2001).

Throughout the paper, the discussion is made more concrete by experiments on a version of the Finnish Constraint Grammar (FINCG), a freely available rule set developed originally for Finnish by Fred Karlsson in Helsinki. The preliminary experiments merely suggest the rough degrees of cardinality in various aspects of processing complexity.

1.2 The Contributions

The current independent work of the author (Yli-Jyrä, 2010) aims at reconceptualizing CG parsing in the framework of finite automata. It describes an efficient parsing procedure where the local ambiguity is summarized with feature vectors. Moreover, a nearly complete CG rule compiler and a partially implemented parser are briefly reported.

The presented approach is in strict contrast to some prior parsers where the local ambiguity domains are represented, tested and reduced by manipulating a linear representation of the set of readings. The linear representation gives rise to CG parsing as transducer sequences (proposed by Lauri Karttunen; see Voutilainen 1994:39, Koskenniemi 1997, Peltonen 2011, Hulden 2011), but is likely to become a bottleneck if syntactic functions and argument structures are provided in the input. In contrast, the current proposal eliminates the distinct syntactic disambiguation rules and compacts the representation of local ambiguity domains.

The current work uses pure finite-state automata that are described, at a high level, using rational sets and series. On the other hand, the paper involves schematic string matching and bidirectional memoization when intersecting automata. These low-level techniques are efficient but differ from standard sequential processing models.

The paper transfers some techniques from the author's prior research on FSIG parsing to the CG framework: (1) *Indexing the transition labels of a template automaton* will compress the implementation of reading subsets (Yli-Jyrä, 1995). (2) The *split languages* of the form $L \subseteq \Sigma^*\Delta\Sigma^*$ (Σ, Δ disjoint alphabets) will represent context conditions (Yli-Jyrä, 2011a). (3) The *position-wise flag diacritics* (Yli-Jyrä, 2011b) will postpone the computation of negation in negative contexts. (4) On-the-fly *inward determinization* (Yli-Jyrä, 2010; Yli-Jyrä, 2011a) will facilitate the iterated computation of product automata. (5) *Preference relations* (Yli-Jyrä, 2007) will enable the ordering of rules and their potential applications. (6) The *infiltration* operation (Sakarovitch, 2009) — a natural implementation of *simple multitape automata* (Yli-Jyrä, 2005) — will elegantly compile the rule conditions. (7) The *string schemas* (Yli-Jyrä, 1995; Yli-Jyrä, 2005; Yli-Jyrä, 2011b) will reduce the length of paths in automata. It is expected that the started CG implementation effort will produce ideas that will reciprocally enrich the FSIG framework.

2 The Primary Representation

The Tokens The natural language sentence – the input of the parser – is preprocessed for parsing by segmenting it into *orthographic words* aka *tokens* $t_1, ..., t_n$. Thus, we obtain e.g. the orthographic words `"<It>"`, `"<rains>"`, `"<.>"` from the sentence "*It rains.*" for the lexical look-up. The resulting segmentation must be unique.

The lexicon is a regular relation that relates the orthographic words with strings consisting of lexemes, morphological labels, syntactic labels and semantic labels – we will call all these symbols naively just *tags* and their strings *readings*.

For every token t_i ($1 \leq i \leq n$), the lexicon provides a set of token-analysis pairs $(t_i, a_{i,j})$ that are linearized to a set of strings $t_i a_{i,j}$ as in (1) — in general, such linearization is not a regular relation, and it would be purer to drop the orthographical word in the analysis. Nevertheless, the set is not converted to a linear string in the current work.

$$\left\{\begin{array}{l}\texttt{"<muuta>" "muu" Q PRON PTV SG}\\ \texttt{"<muuta>" "muuttaa" V IMPV ACT SG2}\\ \texttt{"<muuta>" "muuttaa" V PRES/IMPV ACT NEG}\end{array}\right\} \quad (1)$$

The Cohort Automata The purpose of the CG parser is to reduce excessive readings through removals and selections. By a metaphor, the group of readings for each token is called a *cohort* and the readings manipulated by the operations are called *targets*.

The current parser contains a custom input function for cohorts. This produces deterministic acyclic finite automata $c_1, ..., c_n$, called *cohort automata* (Figure 1), and minimizes them. The cohort automata constitute the *primary representation* of the cohorts.

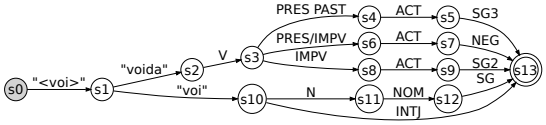


Figure 1: A cohort automaton.

The operators of the CG rules (Tapanainen, 1996) define how cohort automata change if the rules are applied to them. Each operator is a regular relation. To avoid non-termination (Didriksen, 2010), we exclude all rule operators that replace readings, shorten the readings (e.g. by removing `@`-tags), and introduce readings. This leaves such

operators that refine the cohort strictly monotonically: readings are removed with the REMOVE and SELECT operators and new (irreversible) distinctions are introduced with the ADD operator.

The Template Automaton A common experience is that the interface between the grammar and the morphological component gets easily broken if one component is changed. It is thus desirable that CG grammars are augmented with a specification that describes the possible readings by a *template automaton*, a simple positional model (a linear finite automaton) for the tags.

The grammar distinguishes only a finite number of different kinds of tags. In this sense, the tags (the tag types) form a finite set, T. The interface of FINCG needs some 1230 tags, slightly more than are actually used by the rules.

A template automaton for the FINCG is given by a regular expression (2). The special symbol 0 is interpreted as the empty string. This gives an automaton with 1272 transitions (removing the symbol 0 results in 3853 transitions).

```
("<>"|"<ajan>"|...) 0 ("<(.*)ja>"r|...)* 0
("" |"aamu"|...) 0 (DV-JA|...)* 0 (0|DEM|...)
(A|V|...) (0|PRES|...) (0|ACT|...) (0|SG1|...)
(0|CMP|...) (0|ALL|...) (0|SG|...) (0|P-3|...)
   (ko|han|...)*0 (0|@ADVL|@SUBJ|...).        (2)
```

The Dynamic Aspects The expression (2) uses two meta-symbols: "<>" matches any orthographical strings such as "<foo>", and "" matches any lexemes. The tags such as "<(.*)ja>"r are inserted by the input function to the reading strings when the regular expression in it matches the orthographical string such as "<opettaja>"r.

When the parser is used, the conformance of the readings against the model (2) is checked on the fly. The reported anomalies in the lexicon can then be fixed in order to optimize the interoperability between the lexicon and the grammar. In addition, the tags in all readings are indexed with the corresponding source states in the template automaton. The tag DEM, for example, thus becomes DEM_7. There are tags that can correspond to several states — especially if the template automaton is without the symbol 0.

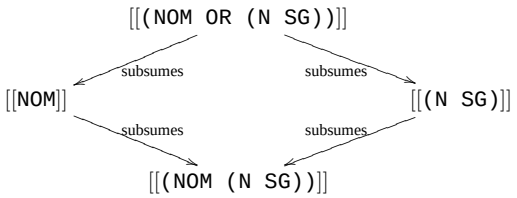


Figure 2: An excerpt of the Boolean lattice.

3 Testing the Cohorts

3.1 The Features of Readings

The CG-2 rules refer to the abstract *features* of complete readings through tags, combined tags, lists, and sets (Tapanainen, 1996) — let us call them *set definitions*. Each set definition α has the denotation $[[\alpha]]$ that is a subset of the readings recognized by the template automaton. Two definitions are *equivalent* if their denotations coincide. The special expression (*) subsumes the whole universe while expression (N SG) subsumes all Singular Noun readings.

In the VISL CG-3 (Didriksen, 2010) system and the current system, the order of tags does not matter i.e. $[[(\text{N SG})]] = [[(\text{SG N})]]$. The denotationally equivalent set definitions form, thus, a Boolean lattice (Figure 2) with denotational union (OR), denotational intersection ( ), and denotational complement (\) operations. The implementation of these operations benefits from the restrictions imposed by the template automaton.

The denotation $S \subseteq T^*$ of every set definition, i.e. a feature f, gives rise to a weighted finite automaton (Sakarovitch, 2009) that recognizes its characteristic series χ_f: $T^* \to \mathbb{N}$ defined by $\chi_f(x)=1$ for all $x \in S$ and $\chi_f(x)=0$ otherwise. We will call this automaton a *feature automaton*. A simple expression, (V), compiles into an automaton with more than 1100 (logical) transitions. To test a feature f against a reading $r \in T^*$, we simply compute $\chi_f(r)$ with the automaton.

When extended to the set of all features F in the grammar, the feature automaton recognizes a series $T^* \to \mathbb{N}^F$ whose coefficients are integer vectors. The easiest implementation of this would be the union of $|F|$ feature automata. The size of this automaton is a severe problem. In the FINCG rule set, the set of tags (T) and the set of features (F) have roughly the same cardinality (respectively: 1133, 1216). Thus, more than a million (logical) transitions are traversed to check all the readings and features in the worst case (consider a cohort

automaton that equals the template automaton). A minimal deterministic automaton is not likely to be any better solution as the deterministic union of all feature automata requires, in the worst case, $O(2^{|F|})$ different final states.

To reduce the complexity of the deterministic union of feature automata, the parser uses a contraction technique (Yli-Jyrä, 1995; Yli-Jyrä, 1997; Roche, 1997a) that hides transitions that are irrelevant to feature recognition. The resulting deterministic automaton will match indexed subsequences in the readings. Given that N and GEN are labels that occur respectively in states 8 and 13 of the template automaton, the set definition (N \GEN) corresponds to a 3-state automaton that recognizes the subsequences

$$\begin{aligned}\mathtt{N}_8(&\mathtt{0}_{13}|\mathtt{ABE}_{13}|\mathtt{ABL}_{13}|\mathtt{ACC}_{13}|\mathtt{ADE}_{13}|\mathtt{ALL}_{13}|\mathtt{CMT}_{13}|\\ &\mathtt{ELA}_{13}|\mathtt{ESS}_{13}|\mathtt{ILL}_{13}|\mathtt{INE}_{13}|\mathtt{INS}_{13}|\mathtt{LAT}_{13}|\mathtt{LOC}_{13}|\\ &\mathtt{MAN}_{13}|\mathtt{NOM}_{13}|\mathtt{PTV}_{13}|\mathtt{TRA}_{13}).\end{aligned} \quad (3)$$

3.2 The Features of Cohorts

In general, the cohorts may mix both readings that satisfy a given feature and readings that do not satisfy it. Therefore, the features become 3-valued at the cohort level: they can be *positive* (+), *negative* (-) or *ambivalent* (?). Testing the cohorts corresponds to computing the function: $2^{T^*} \rightarrow \{+,-,?\}^F$, where the domain 2^{T^*} contains all the possible cohorts and the range $\{+,-,?\}^F$ contains all combinations of the features.

For each cohort, the truth-value of a particular feature f is determined by counting the number of true readings in the domain of χ when it is restricted to the set of readings S in the cohort automaton: the feature automaton is first restricted with the cohort and then the sum of the successful paths is computed e.g. by replacing all the inputs with the empty string. Thus, we compute $\sum_{x \in S} \chi_f(x)$.

In order to get the correct counts, the feature automata must be *path unambiguous* i.e. they assign, at the most, one successful path to each tag sequence. Ensuring an unambiguous automaton for set definitions like (C OR "että") requires special attention, because the features C and "että" are non-exclusive and are separately true in the reading ("<että>" "että" SUB C).

Finally, the integer vectors $\mathbb{N}^F$ are mapped to the vectors of (3-valued) truth values, $\{+,-,?\}^F$. If the count of a feature equals the cardinality of the cohort, the feature is positive +. Otherwise, it is negative - if the cardinality is zero or ambivalent ? in other cases.

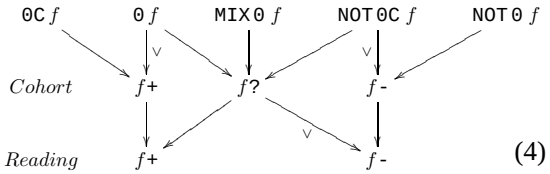
Simple Conditions A typical CG rule specifies three things: an *operation* that tells how a cohort will change if the rule is applied to it, the *context specification* that tells where the rule can be applied, and a *target* that complements the operation and the context specification. The context specification is a conjunction of simple, possibly linked, context conditions (Tapanainen, 1996).

A simple context condition in CG rules is, formally, a 6-tuplet ⟨*origin*, *polarity*, *position*, *mode*, *set definition*, *barrier condition*⟩ where

- the *origin* refers to the target cohort (nothing) or to the hit of the previous positive condition (LINK)
- the *polarity* is either *negative* (NOT), *positive* (nothing), or *ambivalent* (MIX)
- the *position* is either an *absolute position* (@1, @2, ...), a *relative position* (..., -2, -1, 0, 1, 2, ...) with 0 as the origin, or an *unbounded set of relative positions* outwards from the origin (..., *-2, *-1, *0, *1, *2, ...)
- the *mode* is *careful* (C) or *normal* (nothing)
- the *barrier condition* is either none (nothing) or a set α (BARRIER α).

A condition such as ⟨LINK, not, 0, C, N, BARRIER CLB⟩ is written simply as (LINK NOT 0C N BARRIER CLB). The reader is referred to CG manuals for a complete description of context conditions (Karlsson et al., 1995; Tapanainen, 1996).

The combination of polarity and mode tells how the cohort's feature-values are converted back to Boolean truth-values needed in the grammars. This is illustrated in (4).



(4)

The core CG grammar rules — those whose operator is SELECT and REMOVE — specify a set definition called a *target*. A target (NOM) is, in fact, shorthand for condition (MIX 0 NOM). The condition matches cohorts where some readings contain the NOM tag and some do not.

4 The Secondary Representation

We will now see how the conjunction of the contextual conditions is represented in the parser.

4.1 Marking the Potential Applications

The Feature-Value Alphabet Internally, the set definitions are mapped to feature numbers that form the feature alphabet F. The numbers of equivalent set definitions coincide. For example, the current parser assigns the feature numbers 31 and 39 to set definitions $\texttt{(PTV)}_{31}$ and $\texttt{("yhtään" ADV)}_{39}$, respectively. We write `31.PTV` when we emphasize the feature number (the key) rather than the set definition (the legend).

The core FINCG rules mention 1216 distinct features (F) and 1813 feature-value pairs such as (`5.NOM`,`+`) and (`11.SG`,`?`).

The *secondary representation* for the cohorts consists of the feature-value pairs ($P{=}F{\times}\{$`+`, `-`, `?`$\}$) and cohort boundaries ($\bullet$). The pairs in P are written simply as `5.NOM+` and `11.SG?`. The whole sentence is represented by the $\bullet$-marked concatenation of the cohort-wise lists $c'_1, ..., c'_n$ of $|F|$ numerically ordered feature-value pairs.

$$w = c'_1 \bullet c'_2 \bullet ... \bullet c'_n. \tag{5}$$

The Rule Marker Alphabet In order to talk about rules and where they apply, we define a symbol alphabet for the rules. First, the rules are numbered. The rule compiler assigns numbers 343 and 865 to the rules

$$\texttt{SELECT("yhtään" ADV)}_{39} \quad \texttt{IF(NOT}^{*}\texttt{1 (PTV)}_{31}\texttt{);} \tag{343}$$

$$\texttt{SELECT(N SG)}_{821} \quad \texttt{IF(1* (V SG3)}_{820}\ \texttt{BARRIER(CLB)}_{3}\texttt{)} \quad \texttt{(NEGATE*-1(N NOM)}_{715}\texttt{BARRIER(CLB)}_{3}\texttt{).} \tag{865}$$

For every rule, there are two *rule markers* that are used to indicate the satisfied context conditions of the rules. The markers form the set $R = \{$`@1.`r`@`, `@2.`r`@` $\mid r$ is a rule in the grammar $\}$. As to FINCG, there are about 1380 `SELECT`/`REMOVE` rules and 22 (1,6%) of them use both kinds of markers.

The marker `@1.865@` will be used to indicate the cohorts whose contexts satisfy the licensing context conditions of the rule 865. The marker `@2.865@` will be used to indicate the cohorts that satisfy a prohibiting context condition. The prohibiting context conditions are stronger and will prevent the application of the rule even if the licensing context condition is true (Yli-Jyrä, 2011b).

The CG-3 syntax for rule conditions separates the cohort-internal negation (`NOT`) from the global negation (`NEGATE`) (Didriksen, 2010). The first refers to negative features and the second starts a prohibiting condition according to the CG rule syntax. For example, the rule 343 selects the Adverb (`39.ADV`) reading if the next word has no Partitive (`31.PTV`) readings. In contrast, the rule 865 selects the Nominative Singular (`821.(NOM SG)`) reading if the cohort is followed by potential (i.e. positive or ambivalent) 3rd Person Singular Verb (`820.(V SG3)`) within the same clause (`BARRIER (3.CLB)`) unless (`NEGATE`) the cohort is followed by potential Nominative Noun (`715.(N NOM)`) cohort within the same clause.

Split Languages The string w provides a matrix structure against which the licensed and prohibited applications of CG rules are indicated: for each cohort where the rule r is licensed, we produce a copy of w and insert the marker `@1.`r`@` into the end of the cohort's feature-value list in this copy. The marker `@2.`r`@` is inserted in a similar way to other copies that indicate prohibited contexts.

Let $\Sigma = P \cup \{\bullet\}$. A language of the shape $L \subseteq \Sigma^* R \Sigma^*$ is called a *split language*[1]. To facilitate the formal account of the marking, define a mapping $h : \Sigma^* R \Sigma^* \to \Sigma^*$ by $h = \{(vxy, vy) \mid v, y \in \Sigma^*, x \in R\}$. The inverse of the image of w is $h^{-1}(w)$, the language of the possible ways to insert a rule marker into the string w. This language is recognized by an automaton that is very similar to the linear automaton recognizing the string w. The automaton has $O(n|F|)$ states and $O(n|F||R|)$ transitions.

The union of the licensing and prohibiting contexts of each rule r forms a split regular language $C_r \subseteq \Sigma^* R_r \Sigma^*$ where $R_r = \{$`@1.`r`@`, `@2.`r`@`$\}$. For the rule, the marked copies of the string w are obtained as the intersection $W_r = C_r \cap h^{-1}(w)$. The potential applications of all rules are obtained as the union $W = \bigcup_{r \in R} C_r \cap h^{-1}(w)$.

4.2 Controlling the Application Order

The prohibited applications are subtracted from the licensed ones by computing $W' = \{v$`@1.r@`$y \mid v$`@1.r@`$y \in W, v$`@2.r@`$y \notin W\}$. Such preference restrictions can be implemented with a matching

[1] The term was proposed to me by J. Sakarovitch.

method due to Dale Gerdemann and Gertjan van Noord (see Yli-Jyrä, 2007, 2011b). The same method implements application order modes.

The current system is flexible enough to support any standard application mode. Normally, the application of an earlier rule in the grammar is preferred over the later rules. On the other hand, it is psycholinguistically motivated to process the sentence from left to right. By emphasizing the leftmost position, for example, we get the following restriction of W':

$$\begin{aligned} W'' = \{ & v\texttt{@1.r@}y \in W' \mid \\ & \neg\exists u\texttt{@1.r@}z \in W' \text{ s.t. } vy = uz, |u| < |v| \\ & \neg\exists v\texttt{@1.q@}y \in W' \text{ s.t. } q < r\}. \end{aligned} \quad (6)$$

The obtained set W'' contains (at the most) one marked copy w' of the sentence w. The marker in this copy indicates which rule is applicable to which cohort.

When given an m-state deterministic automaton representing the set W, we can compute W' and W'' in $O(m)$ time. It should be noted that the subsets of $\{vxy \mid vy = w, x \in R\}$, where w is the sentence and R is the rule marker alphabet, are always regular languages and their recognizers are minimizable in linear time.

5 The Context Automata

For each rule $r \in R$, the intersection $C_r \cap h^{-1}(w)$ is computed through the well-known product construction (see Sakarovitch, 2009).

The constructed product may have a large number of states: Let the context language C_r be given by an $O(m)$-state deterministic finite automaton. The minimal automaton recognizing the language $h^{-1}(w)$ has $O(n|F|)$ states. The product of these automata has, thus, $O(nm|F|)$ states. When this is repeated for all $r \in |R|$ rules, the total state complexity is $O(nm|R||F|)$.

The product is potentially constructed several times. A pathologically ambiguous cohort can be refined separately by $|R|$ rules if the targets of the rules have disjoint denotations. Therefore, the computation of the intersection is iterated $O(n|R|)$ times in the worst case. This means that the total time complexity of the context testing is $O(n^2m|R|^2|F|)$ in the worst case.

We need optimizations that reduce the effects of (1) the context states, (2) the iterations, (3) the features, and (4) the parallel rules. The following will sketch some important optimization strategies.

5.1 Reducing the Effect of Context States

The Baseline The deterministic automaton recognizing the context language C_r is deterministic up to point where an R-transition is followed. After this point, the product automaton contains, in the worst case, $O(m)$ parallel paths (Figure 3(i)).

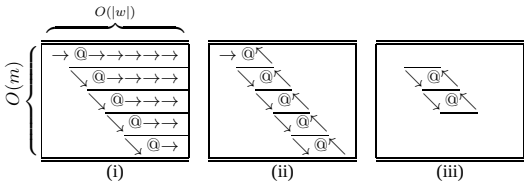


Figure 3: The benefits of inward processing.

Inward Processing It is desirable to find a method that computes the product or it restriction W'' in time that does not depend on the state complexity of the context languages. Three solutions that fulfill this condition are available:

1. The function returning W'' for each w is a has a deterministic realization using *deterministic bimachines* (Skut et al., 2004; Roche, 1994; Peikov, 2006; Hulden, 2011).

2. The context language C_r is recognized by an *inward deterministic automaton (IDA)* (Yli-Jyrä, 2011a). The approach allows for greater flexibility in the application ordering.

3. An IDA can be factorized to possibly smaller left- and right-sequential transducers (Roche, 1997a). Nondeterministic bimachines could be used too (Santean and Yu, 2006).

An *IDA* is a nondeterministic automaton that recognizes a split language $L \subseteq \Sigma^* R \Sigma^*$ and is deterministic for all prefixes in Σ^* and codeterministic for all suffixes in Σ^*. When intersected with the linear automaton representing the language $h^{-1}(w)$, the restricted product approaches the R-transitions deterministically from both sides (Figure 3(ii)). As a result, each of $O(n)$ positions in w corresponds, at the most, to two states in the restricted product (Figure 4). With one IDA per rule, the total time complexity is bounded by $O(n^2|R|^2|F|)$.

State Explosion It would be nice to construct a combined IDA that recognizes the split language $\cup_{r \in R} C_r$. This would reduce the total computation time to $O(n^2|R||F|)$ if we assumed that parallel

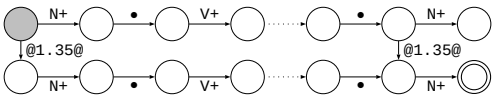


Figure 4: An inward deterministic product.

marker transitions are inserted into the result in $O(1)$ time. However, the whole grammar cannot be combined to an IDA in general. Firstly, there are simple context conditions that require a very large number of IDA states. Secondly, the size of a combined IDA would be $O(m^{|R|})$ if each rule-specific IDA had m states at the most.

5.2 Reducing the Effect of Iterations

Dynamic Programming The product automaton is not constant, but reflects the changes in the sentence w when rules are applied. With inward deterministic contexts, the product is easier to keep up to date. Namely, the previously computed product automaton can be refreshed locally around the changed cohort (Figure 3(iii)).[2] If the distance between the successive refinements is $O(1)$ according to the amortized analysis over the total of $O(n|R|)$ refinements, the local refreshing of the product improves the time complexity to $(n|R|^2|F|)$.

On-the-fly Inward Determinization When computing the intersection $C_r \cap h^{-1}(w)$ with a nondeterministic recognizer for C_r, the result can be determinized easily on-the-fly due to the special structure of the language $h^{-1}(w)$. The result is still virtually deterministic from both sides, allowing for the efficient refreshing of the product after a cohort is refined and local changes in w occur. Under the assumption of $O(1)$ distance between the refinements, the worst case time complexity is now $O(nm|R|^2|F|)$.

5.3 Reducing the Effect of Features

The Path Length Problem A problem with all intersection methods is that they process full-length paths. The feature values in each cohort are read one-by-one, resulting in many states and transitions. Because the length of the string w is $O(n|F|)$, the path length is a practically significant problem.

[2]The dynamics of the optimal refreshing of the product has been studied by the author in a PSC submission (2010).

Contracted Contexts To address the path length problem, the rule compiler of the parser contracts the strings in a context language C into the patterns in a contracted context language C' that retains just the necessary details. In the patterns, the cohort boundaries are retained, while the feature-value symbols in F are retained only where the rule condition refers to them. For example, the condition (`-5 C`) corresponds to the regular language $\bullet^*$`C`$\bullet\bullet\bullet\bullet\bullet$`@1.`$r$`@`$\bullet^*$.

The intersection of the patterns is computed against the image $g(h^{-1}(w))$ where $g \subseteq (\Sigma \cup R)^* \times (\Sigma \cup R)^*$ is a regular relation defined by $g(\epsilon)=\epsilon$, $g(\bullet)=\bullet$, $g(r)=r$, $g(f)=\{f, \epsilon\}$, $g(xy)=g(x)g(y)$, for $r \in R$, $p \in P$, $x, y \in (\Sigma \cup R)^*$.

The language C' is a *valid contraction* of the context language C if

$$C \cap h^{-1}(w) = g^{-1}(C' \cap g(h^{-1}(w))) \cap h^{-1}(w). \quad (7)$$

For CG rules, the valid contractions of context languages can be constructed easily because they do not need to express negations through the absence of features.

Compressed Sentence Automaton The language $g(h^{-1}(w)))$ does not give rise to particularly useful representations of the sentence. Instead, a good representation for the substrings of w is obtained by an n-state sentence automaton where each state contains the loops for the feature-value symbols and are connected with the cohort boundary symbols (Figure 5). Under the intersection $C' \cap g(h^{-1}(w)))$, this is a lossless compression because C' still respects the order in P.

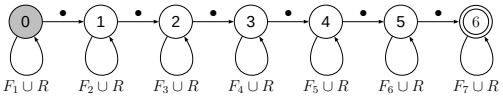


Figure 5: A compressed sentence automaton.

5.4 Reducing the Effect of Rules

The contracted context languages have, rulewise, quite small minimal deterministic recognizers. For example, the rules 343 and 865 give rise to the recognizers in Figure 6.

There are 1377 rules. The sum of the sizes of the recognizers is 10529 states and 16707 arcs. This means, on average, 7.6 states and 12.1 transitions per rule.

A *grammar automaton* recognizes the union of all contracted context languages. For FINCG,

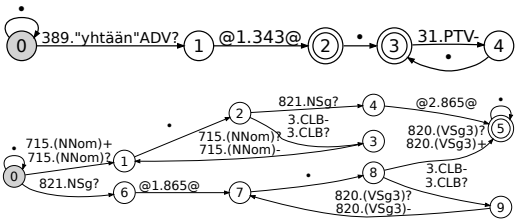


Figure 6: Two contracted context automata.

the obtained minimal deterministic grammar automaton has only 5277 states and 13445 transitions. This saves some space because of the shared states. On average, each rule corresponds to 3.8 states and 9.8 transitions.

A striking thing in this automaton is that 98% of the states have 1 - 7 transitions and three states have 887 - 919 transitions. This suggests that, in the product construction, the computation of the product can be optimized by taking advantage of the size asymmetry inside the product states and the probability of the transitions. If $|F|$ is fixed, the product is computed in $O(ne)$ time where e is the number of states in the grammar automaton.

The total complexity of iterated context testing is bounded by $O(n^2|R|e)$. It is conjectured, however, that the average time complexity of a proper implementation is close to $O(ne \log |R|)$ because (i) the amortized $O(1)$ proximity and refreshing can eliminate the effect of $O(n)$ iterations and (ii) an average cohort is refined by partitioning it into two halves ($\log |R|$ rather than $|R|$).

5.5 Compilation of Contracted Contexts

The Infiltration In formal language theory (Sakarovitch, 2009), the *infiltration* of words $u, v \in \Gamma^*$, denoted with $u \uparrow v$, is defined as the set of words w s.t. subwords u and v cover w completely. More formally, $u \uparrow v$ consists of strings $x_1...x_n \in \Gamma^*$ for which

$$I = \{i_1, ..., i_n\}, i_1 < i_2 < ... < i_n, u = x_{i_1}...x_{i_n},$$
$$J = \{j_1, ..., j_n\}, j_1 < j_2 < ... < j_n, v = x_{j_1}...x_{j_n},$$
$$I \cup J = \{1, ..., n\}.$$

The set $\{\texttt{ABC}, \texttt{ABBC}, \texttt{ABCB}, \texttt{BABC}, \texttt{BACB}, \texttt{BCAB}\} \subseteq \Gamma^*$, for example, is given as $\texttt{AB} \uparrow \texttt{BC}$. The operation extends additively to languages $U, V \subseteq \Gamma^*$ by the definition $U \uparrow V = \{w \in u \uparrow v \,|\, u \in U, v \in V\}$. The infiltration operation is implemented by a state pair construction over automata.

The Synchronization The CG compiler combines conjunctive conditions through the *synchronized infiltration operation* $U \uparrow_{S,<} V$ where $U, V \subseteq \Sigma^*$. The infiltration is restricted in two ways: First, we define a set of *synchronization symbols* S that include the cohort boundary symbol (•), and require that $\{i \mid x_i \in S\} \subseteq I \cap J$ holds for this set. Second, we require that the adjacent letters x_i, x_{i+1} are in a strictly increasing order ($<$) if neither is the cohort boundary symbol (•). Feature symbols `SG+`, `SG−` and `SG?`, for example, are incomparable letters of Σ, which means that they cannot be adjacent with each other.

The Anchors The relative conditions are anchored to the target cohort by a rule marker. Therefore, rule markers `@1.`r`@`, `@2.`r`@` $\in R$ are called *anchors*. The linked conditions use other anchors whose shape is `@LINK.`n`@` ($n = 1, 2, ...$). Under the synchronized infiltration, the set of synchronization symbols includes the anchors shared by both U and V. The internal anchors are suppressed after they have been used as synchronization symbols. The application of synchronized infiltration to linked conditions is illustrated in Figure 7.

6 The Implementation

The current rule compiler has been created by adapting the skeleton of the Foma tool (Hulden, 2009) to the purposes. The extensions include the infiltration operation, the symbol tables and other essential compiler logic. The compiler is integrated into the parser, whose data structures and algorithms were written from the scratch in order to optimize the computations on cohort automata. Some parts of the system are still under construction. Therefore, experiments on complete parsing are not yet available.

The contracted grammar automaton of 1380 rules is constructed in 20 seconds by the rule compiler. In the parser, about 110 000 cohort automata were read, minimized and mapped to the integer vectors of 1216 features in 1 second (2.2-Ghz Core-2-Duo laptop).

7 Evaluation

The presented parser design has many advantages that concern the space and time requirements and possible extensions. On the other hand, having two finite-state representations of the sentence is

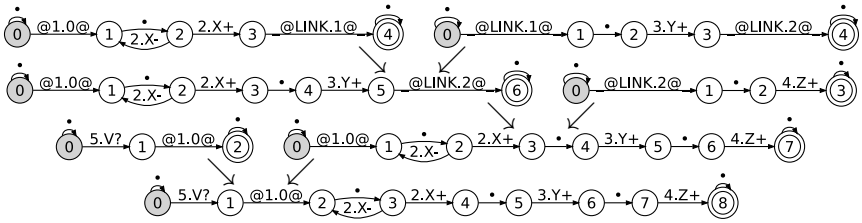


Figure 7: The contracted context for [SELECT (V) ((*1C X) (LINK 1C Y) (LINK 1C Z))]$_0$ is computed with synchronized infiltration and by removing the intermediate link anchors.

clearly a conceptual complication, requiring a special parsing algorithm that ties the representations to each other.

Space Advantages The automaton representation of the grammar is *close to the original grammar size*, fitting easily into cache memories. The product of the grammar automaton and the sentence is *small* and easy to compute.

Speed Advantages The current design is a step towards high-speed CG parsing. This is argued by the following points: (i) The *low space complexity* sets better lower bounds for the time complexity. (ii) The iterations take advantage of the *prior contextual tests*. (iii) The *cohort automata* compress ambiguity. (iv) The *contractions* optimize the constructions. (v) The rules *share* various common parts, which makes the parallel testing of contexts faster. (vi) The combined contextual tests are *compiled* for each rule.

Possible Extensions The current design increases the flexibility of the CG rules: (i) The *application mode* can be altered easily. (ii) *Coordinated disambiguation rules* can be implemented for syntagmatic patterns. (iii) The context conditions can be arranged to *levels of exceptions* (@1.r@, @2.r@, @3.r@,...). (iv) The rule *formalism* can be extended. (v) The borderline between morphological and syntactic rules can be removed by *lexicalizing* the intermediate mapping.

8 Conclusions

The paper has described a nonconventional CG parser architecture using finite-state methods. In the approach, the list representation of the readings is replaced with a cohort automaton and feature vectors. The readings and contexts are tested with contracted patterns. Iterated testing is optimized with inward processing. The presented architecture and its prototype are expected to lead to an efficient and flexible parser and enrich the related research.

Acknowledgments

The work has been supported by the grant #128536 “Open and Language Independent Automata-Based Resource Production Methods for Common Language Research Infrastructure” of the Academy of Finland. I am also indebted to A. Voutilainen, F. Karlsson, K. Koskenniemi, K. Lindén, and T. Trosterud for long-time interest, PSC 2010 reviewers for suggestions, CG 2011 participants for helpful responses in Riga, and for M. Hulden for discussions in Blois.

References

Steven Abney. 1991. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers.

Salah Aït-Mokhtar and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proc. 5th ANLP, the Conference on Applied Natural Language Processing*, pages 72–79, Washington, DC.

G. Edward Barton, Jr. 1986. Constraint propagation in Kimmo systems. In *Proc. 24th ACL*, pages 42–52, New York, NY, July 10-13. The Association for Computational Linguistics (ACL), Stroudsburg, PA.

Glenn David Blank. 1989. A finite and real-time processor for natural language. *Communications of the ACM*, 32(10):1174–1189, October.

Henning Bordihn, Henning Fernau, Markus Holzer, Vincenzo Manca, and Carlos Martín-Vide. 2006. Iterated sequential transducers as language generating devices. *Theoretical Computer Science*, 369:67–81.

Eric Brill. 1992. A simple rule-based part of speech tagger. In *Proc. 3rd ANLP*, Trento, Italy

Kenneth Ward Church. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. 2nd ANLP*, pages 136–143, Austin, TX.

Tino Didriksen. 2010. Constraint Grammar Manual: 3rd version of the CG formalism variant. GrammarSoft Aps, Denmark.

Jorge Graña, Gloria Andrade, and Jesús Vilares. 2003. Compilation of constraint-based contextual rules for part-of-speech tagging into finite state transducers. In *Proc. 7th CIAA, the Conference on Implementation and Application of Automata*, pages 128–137, Springer-Verlag, Berlin, Germany.

Gregory Grefenstette. 1999. Light parsing as finite state filtering. In András Kornai, editor, *Extended finite state models of language*, pages 86–94. Cambridge University Press, New York, NY.

Maurice Gross. 1968. *Grammaire transformationnelle du francais*, volume 1, Syntaxe du verbe. Larousse, Paris, France.

David A. Huffman. 1971. Impossible Objects as Nonsense Sentences. *Machine Intelligence*, 6:295–323.

Mans Hulden. 2009. Foma: a finite-state compiler and library. In *Proc. Demonstrations Session at the 12th EACL*, pages 29–32, Athens, Greece. ACL, Stroudsburg, PA.

Mans Hulden. 2011. Constraint grammar parsing with left and right sequential finite transducers. To appear in *Proc. 9th FSMNLP*, Blois, France. ACL, Stroudsburg, PA.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Number 3 in Monographs on linguistic analysis. Mouton, The Hague.

Aravind K. Joshi and Phil Hopely. 1996. A parser from antiquity. *Natural Language Engineering*, 2(2):291–294.

Aravind K. Joshi and Leon S. Levy. 1982. Phrase structure trees bear more fruit than you would have thought. *American Journal of Computational Linguistics*, 8(1):1–11.

Tomasz Jurdziński, Friedrich Otto, František Mráz, and Martin Plátek. 2005. Deterministic two-way restarting automata and Marcus contextual grammars. *Fundamenta Informaticae*, 64(1–4):217–228.

Fred Karlsson, Atro Voutilainen, Juha Heikkiä, and Arto Anttila, editors. 1995. *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*, volume 4 of *Natural Language Processing*. Mouton de Gruyter, Berlin and New York.

Fred Karlsson. 1990. Constraint Grammar as a framework for parsing unrestricted text. In H. Karlgren, editor, *Proc. 13th COLING*, volume 3, pages 168–173, Helsinki, Finland. International Committee on Computational Linguistics (ICCL).

Lauri Karttunen. 1997. The replace operator. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, chapter 4, pages 117–147. A Bradford Book, the MIT Press, Cambridge, MA, USA.

Lauri Karttunen. 1998. The proper treatment of optimality in computational phonology. In *Proc. 2nd FSMNLP*, pages 1-12, Bilkent University, Ankara, Turkey.

Kimmo Koskenniemi, Pasi Tapanainen, and Atro Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *Proc. 14th COLING*, volume I, pages 156–162, Nantes, France. International Committee on Computational Linguistics (ICCL).

Kimmo Koskenniemi. 1983. *Two-level morphology: a general computational model for word-form recognition and production*. Ph.D. thesis, number 11 in Publications of the Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki, Finland.

Kimmo Koskenniemi. 1997. Representations and finite-state components in natural language. In Emmanuel Roche and Yves Schabes, editors, *Finite-state language processing*, chapter 3, pages 99–116. A Bradford Book, The MIT Press, Cambridge, MA.

Steven Krauwer and Louis des Tombe. 1981. Transducers and grammars as theories of language. *Theoretical Linguistics*, 8:173–202.

Torbjörn Lager and Joakim Nivre. 2001. Part of speech tagging from a logical point of view. In P. de Groote, G. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 212–227. Springer-Verlag, Berlin, Germany.

Torbjörn Lager. 2001. Transformation-based learning of rules for constraint grammar tagging. Presented at the 13th Nordic Conference in Computational Linguistics, NODALIDA, Uppsala, Sweden, May 21-22.

Nikolaj Lindberg and Martin Eineborg. 1998. Learning constraint grammar-style disambiguation rules using inductive logic programming. In *Proc. 36th ACL / 17th COLING, Montréal, Quebec, Canada, August 10-14*, volume 2, pages 775–779. ACL, Stroudsburg, PA.

Mitchell P. Marcus. 1980. *A Theory of Syntactic Recognition for Natural Language*. The Series in Artificial Intelligence. The MIT Press, Cambridge, MA.

Hiroshi Maruyama. 1990. Structural disambiguation with constraint propagation. In *Proc. 28th ACL*, pages 31–38, Pittsburgh, PA. ACL, Stroudsburg, PA.

Denis Maurel. 1989. *Reconaissance de sé quences de mots par automates. Adverbes de date*. Ph.D. thesis, Université Paris 7, Paris, France.

Mehryar Mohri. 1994. Syntactic analysis by local grammars automata: an efficient algorithm. In D. K. Kiefer, G. Kiss, and J. Pajzs, editors, *Proc. International Conference on Computational Lexicography (COMPLEX 94)*, pages 179–191, Budapest, Hungary.

Kemal Oflazer and Gökhan Tür. 1997. Morphological disambiguation by voting constraints. In *Proc. 35th ACL / 8th EACL*, pages 222–229, Madrid, Spain. ACL, Stroudsburg, PA.

Ivan Petrov Peikov. 2006. Direct construction of a bimachine for context-sensitive rewrite rule. Master's thesis, Sofia University St. Kliment Ohridski, Faculty of Mathematics and Computer Science, Department of Mathematical Logic and Applications, Sofia.

Janne Peltonen. 2011. Rajoitekielioppien toteutuksesta äärellistilaisin menetelmin. Master's thesis, University of Helsinki, Department of Modern Languages, Helsinki.

Paul Stanley Peters and Robert W. Ritchie. 1969. Context sensitive immediate constituent analysis — context-free languages revisited. In *Proc. ACM Symposium on Theory of Computing*, pages 1–8, Marina del Rey, California, May 5–7.

Martin Plátek, Markéta Lopatková, and Karel Oliva. 2003. Restarting automata: motivations and applications. In M. Holzer, editor, *Workshop Petrinetze und 13. Theorietag Automaten und Formale Sprachen*, pages 90–96, Institut für Informatik, Technische Universität München, München, Germany.

Emmanuel Roche. 1994. Two parsing algorithms by means of finite state transducers. In *Proc. 20th COLING*, volume 1, pages 431–435, Kyoto, Japan. International Committee on Computational Linguistics (ICCL).

Emmanuel Roche. 1997a. Compact factorization of finite-state transducers and finite-state automata. *Nordic Journal of Computing*, 4(2):187–216.

Emmanuel Roche. 1997b. Parsing with finite-state transducers. In Emmanuel Roche and Yves Schabes, editors, *Finite-state language processing*, chapter 8, pages 241–281. A Bradford Book, the MIT Press, Cambridge, MA.

Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Cambridge University Press, Cambridge, NY.

Nicolae Santean and Sheng Yu. 2006. On weakly ambiguous finite transducers. In O.H. Ibarra and Z. Dang, editors, *DLT 2006*, volume 4036 of *LNCS*, pages 156–167. Springer-Verlag, Berlin, Germany.

Wojciech Skut, Stefan Ulrich, and Kathrine Hammervold. 2004. A bimachine compiler for ranked tagging rules. In *Proc. 20th COLING*, Geneva, Switzerland. International Committee on Computational Linguistics (ICCL).

Pasi Tapanainen. 1996. *The Constraint Grammar Parser CG-2*. Number 27 in Publications of the Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki, Finland.

Pasi Tapanainen. 1999. *Parsing in two frameworks: finite-state and functional dependency grammar*. Ph.D. thesis. Department of General Linguistics, University of Helsinki, Finland.

Atro Voutilainen. 1994. *Three studies of grammar-based surface parsing of unrestricted English text*. Ph.D. thesis, number 24 in Publications of the Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki, Finland.

William A. Woods. 1970. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):71–87. Association for Computing Machinery (ACM).

Anssi Yli-Jyrä. 1995. Schematic finite-state intersection parsing. In Kimmo Koskenniemi, editor, *Short Papers Presented at the 10th Nordic Conference of Computational Linguistics (NODALIDA-95)*, pages 95–103, Helsinki, Finland, 29–30 May.

Anssi Yli-Jyrä. 1997. Menetelmiä äärellisiin automaatteihin perustuvan lauseenjäsennyksen tehostamiseksi. Master's thesis, Department of General Linguistics, University of Helsinki, Helsinki, Finland.

Anssi Yli-Jyrä. 2005. *Contributions to the Theory of Finite-State Based Grammars*. Ph.D. thesis, number 38 in Publications of the Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki, Finland.

Anssi Yli-Jyrä. 2007. Transducers from parallel replacement rules and modes with generalized lenient composition. In Thomas Hanneforth and Kay-Michael Würzner, editors, *Finite-State Methods and Natural Language Processing, 6th FSMNLP, Revised Papers*, pages 197–212, Potsdam University Press, Potsdam, Germany.

Anssi Yli-Jyrä. 2010. Efficient context-sensitive rewriting with inward deterministic transducers. A submitted manuscript.

Anssi Yli-Jyrä. 2011a. Compiling simple context restrictions with nondeterministic automata. To appear in *Proc. 9th FSMNLP*, Blois, France. ACL, Stroudsburg, PA.

Anssi Yli-Jyrä. 2011b. Explorations on positionwise flag diacritics in finite-state morphology. In Bolette Sandford Pedersen, Gunta Nešpore, and Inguna Skadin, editors, *NODALIDA 2011 Conference Proceedings*, pages 262–269, Riga, Latvia.